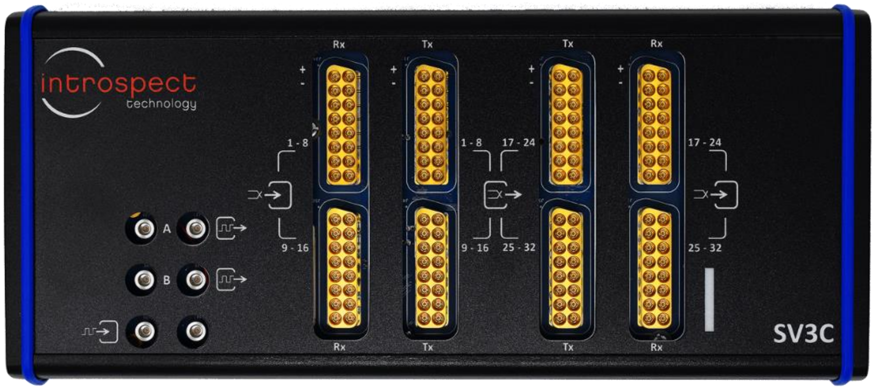
TECHNICAL INSIGHTS

An introduction to the Introspect Vector Blaster



Published in Canada on January 13, 2020

INTROSPECT.CA

Table of Contents

Introduction

Each SV3C Personalized SerDes Tester houses a powerful microcontroller with a wide bidirectional GPIO interface. Introspect Technology has developed a flexible, extremely easy-to-use interface for controlling these GPIOs and has fully integrated the control into our Introspect ESP software. This interface is called the "IESP Vector Blaster".

Benefits and Applications

The IESP Vector Blaster provides a simple mechanism for GPIO control, achieved via an integration of low-level vector files and high-level Python programming, providing the best of two programming worlds. Rapid development and quick turn-around is possible for applications such as:

- implementation of flags and triggers for direct communication with a DUT
- implementation of 32 or 64 bit wide bidirectional buses for communicating with a DUT
- implementation of low-speed interfaces such as I2C

Key Specifications and Features

- Total GPIO pins available: 108
- Maximum Data Rate (per pin): 25 MHz
- Signal standard: 2.5 V LVCMOS
- Seamless integration of IESP Vector Blaster feature set into existing Introspect ESP Software applications

The format for vector files is provided in the next section, followed by an overview of available Python classes for full script automaton. A pinout from the SV3C family of products is also provided.

Vector Blaster Input File Format

The vector patterns are created by the user as a text file describing a list of vectors. A sample of the required vector file format is below:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX1 R10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcXXXXXXXX R5
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc00000000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc11111111 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcXXXXXXXX R100
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcHHHHLLLL R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcLLLLLHHH R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc00000000 R1
```

The first word of each row is the **vector**, followed by a repeat count preceded by the letter **R**.

Each entry in the **vector** can have the following states:

Pin State '0': Drive 0 for 1 UI
Pin State '1': Drive 1 for 1 UI
Pin State 'c': Drive 0 for 1st half of UI, 1 for 2nd half of UI
Pin State 'C': Drive 1 for 1st half of UI, 0 for 2nd half of UI
Pin State ‘X': Set Z (high impedance) for 1 UI
Pin State 'L': Set Z (high impedance) for 1 UI, check if input == 0 at beginning of UI
Pin State 'H': Set Z (high impedance) for 1 UI, check if input == 1 at beginning of UI

Each entry in the vector specifies a repeat count, **R**, that can range from 1 to 65536. The vectors play at a rate of up to 25 MHz (clock period = UI = 40 ns). In a case where a repeat count is set to value of 5, that vector will be repeated 5 times (the equivalent of 5 UI).

Vector pin order is from least significant on the right to most significant on the left. That is, if a 32-bit bus is implemented using bits 31:0 of the IESP vector blaster, and a “walking ones” test was implemented, starting from bit[0] and ending with bit[31], the test vector would be written as shown in the example vector file on the following page.

Example Vector File:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000000001 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000000010 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000000100 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000001000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000010000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00000000000000000000000000100000 R1
#
# full vector set abbreviated for this document
#
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX00100000000000000000000000000000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX01000000000000000000000000000000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX10000000000000000000000000000000 R1
```

NOTE

- Even for a 32 bit implementation (as highlighted in bold) the entire vector must be 108 bits long
- Pins hold their last output state (0, 1 or high impedance) following the end of the execution of a set of vectors

The “Compare” functions allow the IESP Vector Blaster to record the received GPIO states. See the example vector file below:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX1 R10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcXXXXXXXX R5
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc00000000 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc11111111 R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcXXXXXXXX R100
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcHHHHLLLL R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcLLLLLHHH R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc00000000 R1
```

When the vector pattern plays, every vector that contains pins at 'L' or 'H' will check whether those pins are at logic 0 or 1 respectively. If **every** pin comparison in the vector is successful, a result of “1” is stored for that vector, otherwise a result of “0” is stored. When the vector pattern is over, there will be one result bit for each vector that contains at least 1 pin at 'L' or 'H'.

NOTE

- Repeat counts must be set to one for each vector containing a comparison

Python Functions for the Introspect ESP Software

A subset of useful Python functions for use with the IESP Vector Blaster are described below. The complete set of available commands and complete descriptions are available from the Introspect ESP Software GUI via the pull-down Help Menu (Help -> low-level IESP functions…):

sendVectorsFromFile(vectorFilePath)

This function compiles an ASCII-format vector file as described above into the binary form required by the IESP hardware and sends the vector data to the IESP hardware.

Example of use:

```
info = iesp.sendVectorsFromFile(vectorFilePath)
if info is not None:
    (numVectors, numComparisons) = info
```

setNumVectors(numVectors)

Set the number of vectors to be executed. This is only needed if you want to execute a subset of the vectors that were sent, since '*sendVectorsFromFile()*' automatically sets the execution length to all the vectors that were in the file.

Example of use:

```
iesp.setNumVectors(numVectors)
```

startVectorExecution()

This function starts executing the loaded vectors.

Example of use:

```
iesp.startVectorExecution()
```

stepVector()

Execute the next vector if execution paused partway through the loaded vectors.

Example of use:

```
iesp.stepVector()
```

resumeVectorExecution()

Resume executing to the end of the vector list.

Example of use:

```
iesp.resumeVectorExecution()
```

getVectorCompareResults()

This function reads the vector compare results and returns a string (of 0's and 1's) giving the results. If all the vectors in the vector file are executed, the number of bits (number of results of comparisons) should be equal to the 'numComparisons' returned from 'sendVectorsFromFile()'.

Example of use:

```
bitStr = iesp.getVectorCompareResults()
```

getVectorPinState(pin)

Get the current state of a specified vector pin. A 'pin' is an integer specifying a vector pin and should be in the range 0-107. The return value will be either "0" for low or "1" for high.

Example of use:

```
pin = 0
pinState = iesp.getVectorPinState(pin)
```

getVectorPassFail()

Return a Boolean indicating whether the vector execution passed (True) or failed (False). A full vector execution is considered to have passed if each bit of the vector comparison result is "1".

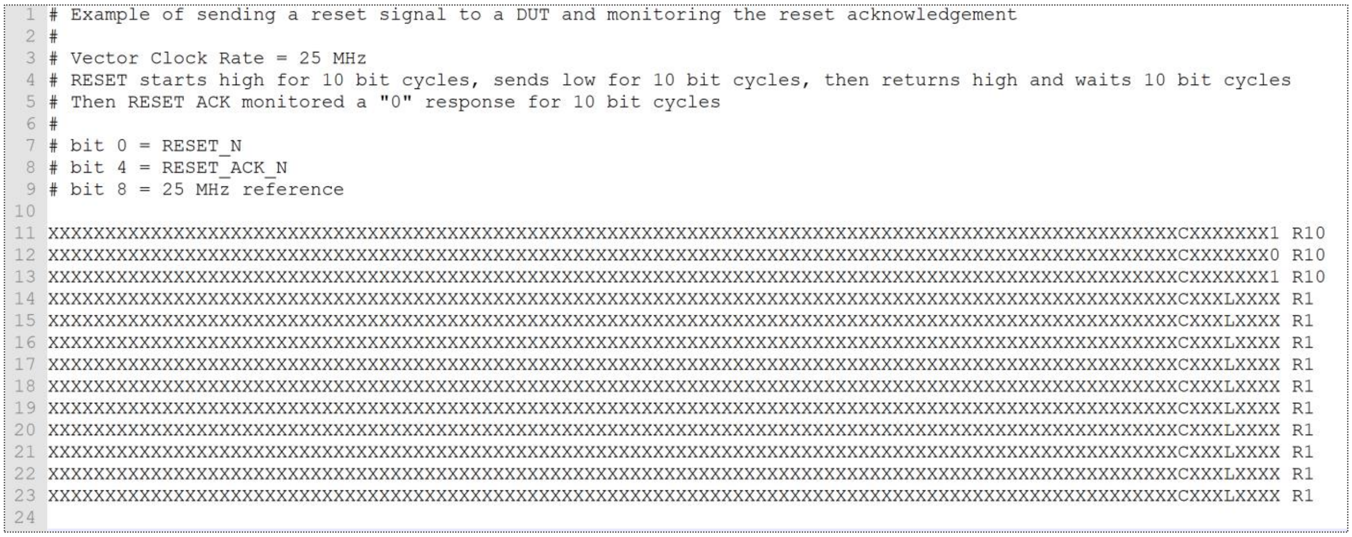
Example of use:

status = iesp.getVectorPassFail()

Example: Sending a Reset Signal to a DUT

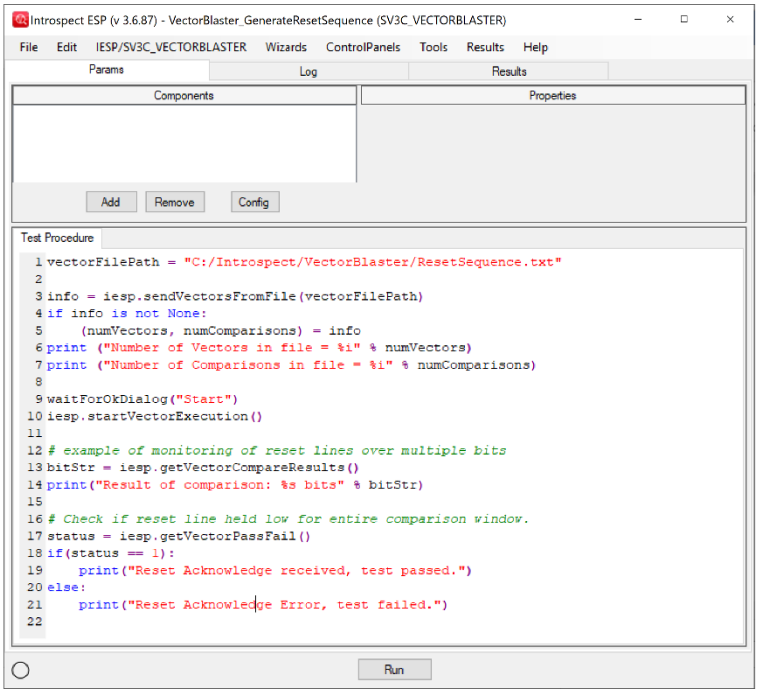
Below is an example which demonstrates a simple application of the IESP Vector Blaster. The vector file consists of a **clock signal**, a **reset** pin (output) and a **reset acknowledge** pin (input), as shown. The following page shows the python-level implementation in the Introspect GUI.

Example Vector File:



```
# Example of sending a reset signal to a DUT and monitoring the reset acknowledgement
#
# Vector Clock Rate = 25 MHz
# RESET starts high for 10 bit cycles, sends low for 10 bit cycles, then returns high and waits 10 bit cycles
# Then RESET ACK monitored a "0" response for 10 bit cycles
#
# bit 0 = RESET_N
# bit 4 = RESET_ACK_N
# bit 8 = 25 MHz reference

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXXXXX1 R10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXXXXX0 R10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXXXXX1 R10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXCXXXLXXXX R1
```



```
vectorFilePath = "C:/Introspect/VectorBlaster/ResetSequence.txt"

info = iesp.sendVectorsFromFile(vectorFilePath)
if info is not None:
    (numVectors, numComparisons) = info
print ("Number of Vectors in file = %i" % numVectors)
print ("Number of Comparisons in file = %i" % numComparisons)

waitForOkDialog("Start")
iesp.startVectorExecution()

# example of monitoring of reset lines over multiple bits
bitStr = iesp.getVectorCompareResults()
print("Result of comparison: %s bits" % bitStr)

# Check if reset line held low for entire comparison window.
status = iesp.getVectorPassFail()
if(status == 1):
    print("Reset Acknowledge received, test passed.")
else:
    print("Reset Acknowledge Error, test failed.")

```

For a test case where the “reset acknowledge” remained low only for 4 bits within the desired window, the following output log messages would be generated:

Output log file:

Components used by Test Procedure: []
Number of Vectors in file = 13
Number of Comparisons in file = 10
Result of comparison: 0000111100 bits
Reset Acknowledge Error, test failed.

For the case where the Reset Acknowledge remained low for all comparison bits within the desired window, the following output log messages would be generated:

Output log file:

```
Components used by Test Procedure: []
Number of Vectors in file = 13
Number of Comparisons in file = 10
Result of comparison: 1111111111 bits
Reset Acknowledge received, test passed.
```

Connectors and Pinout Specifications

The 108 GPIO pins on SV3C devices may be accessed via the high-density Samtec Searay connector on the left side of the module, as shown in the figure on the left below. The pin orientation of the Searay connector the SV3C itself is shown below on the right.

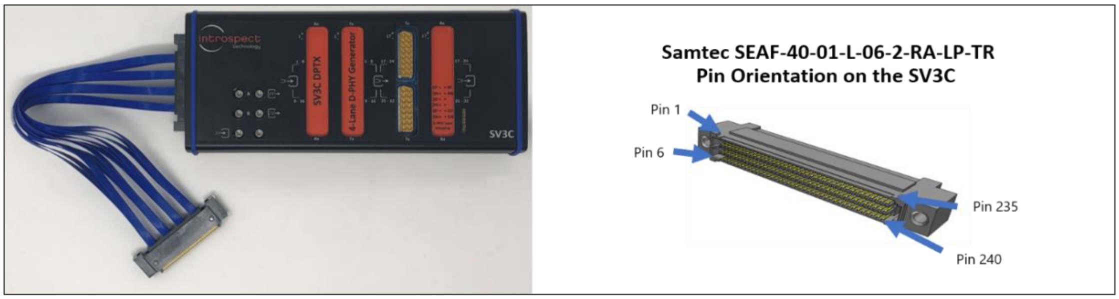
The Samtec part number numbers for connecting to the Vector Blaster interface are as follows:

SV3C Connector: Samtec SEAF-40-01-L-06-2-RA-LP-TR

Connection Cable: Samtec SEAC-040-06-12.0-TU-TU

Recommended customer-side connectors:

Samtec SEAF-40-01-L-06-2-RA-LP-TR (right angle) or SEAF-40-05.0-L-06-2-A-K-TR (straight)



For the complete Searay pinout, please contact Introspect Technology.

Revision Number	History	Date
1.0	Document Release	January 13, 2020

The information in this document is subject to change without notice and should not be construed as a commitment by Introspect Technology. While reasonable precautions have been taken, Introspect Technology assumes no responsibility for any errors that may appear in this document.

Published in Canada on January 13, 2020

INTROSPECT.CA